File Handling

With all our programs so far, there has been a very fundamental limitation: all data accepted is held only for as long as the program remains active. As soon as the program finishes execution, any data that has been entered and the results of
processing such data are thrown away. Of course, for very many real-life applications (banking, stock control, financial accounting, etc.), this limitation is simply not realistic. These applications demand **persistent** data storage. That is to say, data must be maintained in a permanent state, such that it is available for subsequent further processing. The most common way of providing such persistent storage is to use disc files. Java provides such a facility, with the access to such files being either **serial** or **random**.

Serial access files are files in which data is stored in physically adjacent locations, often in no particular logical order, with each new item of data being added to the end of the file.
Class *File* is contained within package *java.io*, so this package should be imported into any file-handling program. Before J2SE 5.0, it was necessary to wrap a *BufferedReader* object around a *FileReader* object in order to read from a file. Likewise, it was necessary to wrap a *PrintWriter* object around a *FileWriter* object in order to write to the file. Now we can wrap a *Scanner* object around a *File* object for input and a *PrintWriter* object around a *File* object for output. (The *PrintWriter* class is also within package *java.io*.)

Method 1

Examples

(i) Scanner input = new Scanner(new File ("inFile.txt"));
(ii) PrintWriter output = new PrintWriter(new File("outFile.txt"));

Method 2

Examples

(i) File inFile = new File("inFile.txt");

Scanner input = new Scanner(inFile);

(ii) File outFile = new File("outFile.txt");
PrintWriter output = new PrintWriter(outFile);

We can then make use of methods *next, nextLine*, *nextInt*, *nextFloat*, ... for input and methods *print* and *println* for output. Examples (using objects *input* and *output*, as declared above)

(i) String item = input.next();
(ii) output.println("Test output");
(iii) int number = input.nextInt();

Example 1: Writes a single line of output to a text file.

```
import java.io.*;


public class File1 {

   public static void main(String[]
args) throws IOException{
```

```
        File s= new File("a.txt");
        PrintWriter output =new
PrintWriter(s );

        output.println ("Bye");

        output.close();

    }

}
```

If the file already existed, its initial contents will have been overwritten.To over come this problem FileWriter class is used

For example:
FileWriter addFile = new FileWriter("data.txt", true);

In order to send output to the file, a *PrintWriter* would then be wrapped around the *FileWriter*:

PrintWriter output = new PrintWriter(addFile);

Example 2: Using File writer class

```
import java.io.*;
public class File2 {

    public static void main(String[] args)
throws IOException {
```

```
        // TODO Auto-generated method stub

        FileWriter fw =new
FileWriter("c.txt",true);
        PrintWriter output1 =new
PrintWriter(fw);
                output1.println("bi");
                 output1.close();

    }

}
```

Example 3: To read the file name from the keyboard and write contents to it.

```
import java.io.*;

import java.util.*;

public class File3 {

    public static void main(String[] args) throws
IOException {

        String fname;
```

```
            int marks;
            Scanner ip=new Scanner(System.in);
            System.out.println("enter a file name");
            fname= ip.nextLine();
            FileWriter fw= new FileWriter(fname);
            PrintWriter op= new PrintWriter(fw);
            System.out.println("enter the 10 marks");
            for(int i=0;i<10; i++)
            {
                  marks= ip.nextInt();
                  op.println(marks);
                  op.flush();
            }
              op.close();
      }
}
```

Example 4: Program to demonstrate to read marks from the file and find the average of marks.

```
import java.io.*;
import java.util.*;
public class File4 {
        public static void main(String[] args) throws
FileNotFoundException
        {
            float marks, total=0, count=0;
            File f= new File("rs.txt");
            Scanner ip =new Scanner(f);
            while(ip.hasNext())
           {
                  marks= ip.nextInt();
                   total= total+marks;
                   count++;
           }
        System.out.println("Average  " +(total/count));
          ip.close();
```

```
        }


}
```

File Methods

Class *File* has a large number of methods, the most important of which are shown below.

• ***boolean canRead()***
Returns *true* if file is readable and *false* otherwise.
• ***boolean canWrite()***
Returns *true* if file is writeable and *false* otherwise.
• ***boolean delete()***
Deletes file and returns *true/false* for success/failure.
• ***boolean exists()***
Returns *true* if file exists and *false* otherwise.
• ***String getName()***
Returns name of file.
• ***boolean isDirectory()***
Returns *true* if object is a directory/folder and *false* otherwise.
(Note that *File* objects can refer to ordinary files or to directories.)
• ***boolean isFile()***
Returns *true* if object is a file and *false* otherwise.
• ***long length()***
Returns length of file in bytes.

• ***String[] list()***
If object is a directory, array holding names of files within directory is returned.
• ***File[] listFiles()***
Similar to previous method, but returns array of *File* objects.
• ***boolean mkdir()***
Creates directory with name of current *File* object.
Return value indicates success/failure.

Program to demonstrate the various methods of File class

```
import java.io.*;
import java.util.*;

public class File8
{

	public static void main(String[] args)
	{

		String filename;
		Scanner input = new Scanner(System.in);

		System.out.println("Enter name of file/directory ");

		System.out.println("or press <Enter> to quit: ");

		filename = input.nextLine();

		while (!filename.equals("")) //Not <Enter> key.
		{
		    File fileDir = new File(filename);

		    if (!fileDir.exists())
		    {
		      System.out.println(filename + " does not exist!");
		       break; //Get out of loop.
		    }

		    System.out.print(filename + " is a ");

		    if (fileDir.isFile())
		       System.out.println("file.");
		else
		  System.out.println("directory.");



		     if (fileDir.canRead())
		       System.out.print("readable. ");
		     else
		       System.out.println("not");
```

```
                if (fileDir.canWrite())
                  System.out.print("writeable. ");
                else
                 System.out.println("not");

               if (fileDir.isDirectory())
               {
                  System.out.println("Contents:");
                  String[] fileList = fileDir.list();
            //Now display list of files in
                 //directory...

                for (int i=0;i<fileList.length;i++)
                System.out.println(" "+ fileList[i]);

            }

            else
            {
              System.out.print("Size of file: ");
              System.out.println(fileDir.length() + " bytes.");
            }

               System.out.print("\n\nEnter name of next file/directory ");
               System.out.print("or press <Enter> to quit: ");
               filename = input.nextLine();
         }

            input.close();
      }
}
```

Command Line Parameters

Program to demonstrate copying the content of source file into destination file using command line parameter.

```
import java.io.*;

import java.util.Scanner;

public class File7

{

        public static void main(String[] args) throws IOException

{

                //First check that 2 file names have been supplied...
```

```
        if (args.length < 2)
        {
        System.out.println("You must supply TWO file names.");
        System.out.println("Syntax:");
        System.out.println(" java Copy <source> <destination>");
        return;
        }
        Scanner source = new Scanner(new File(args[0]));
        PrintWriter destination =new PrintWriter(new File(args[1]));
        String input;
        while (source.hasNext())
        {
        input = source.nextLine();
        destination.println(input);
        }
        source.close();
        destination.close();
    }
}
```

Random Access Files

Serial access files are simple to handle and are quite widely used in small-scale applications or as a means of providing temporary storage in larger-scale applications. However, they do have two distinct disadvantages, as noted below.
(i) We can't go directly to a specific record. In order to access a particular record, it is necessary to physically read past all the preceding records. For applications containing thousands of records, this is simply not feasible.
(ii) It is not possible to add or modify records within an existing file. (The whole file would have to be re-created!) Random access files (probably more meaningfully called **direct access** files) overcome both of these problems, but do have some disadvantages of their own...
(i) In common usage, all the (logical) records in a particular file must be of the

same length.
(ii) Again in common usage, a given string field must be of the same length for all records on the file.
(iii) Numeric data is not in human-readable form.

Program to demonstrate Random Access File (Assume three fields)

```
import java.io.*;

public class Ran
{
    private static final int REC_SIZE=42;

       private static final int NAME_SIZE=15;

       private static RandomAccessFile ranAct;

       public static void main(String[] args) throws IOException
       {
              ranAct= new RandomAccessFile("account3.dat", "rw");
              writeRecord(1000,"rahul",1000);
              writeRecord(2000,"rohith",5000);
              writeRecord(3000,"rakesh",6000);

              showRecords();

       }

       public static void writeRecord(long Actno, String name,float balance ) throws
IOException
       {
              long filepos= ranAct.length();

              ranAct.seek(filepos);
              ranAct.writeLong(Actno);

              writeString(name, NAME_SIZE);

              ranAct.writeFloat(balance);

       }


       public static void writeString(String text, int fixedSize) throws IOException
       {

              int len= text.length();

              if(len<fixedSize)
              {
                     ranAct.writeChars(text);
```

```
                for(int i=len; i<fixedSize; i++)
                {

                ranAct.writeChar(' ');
                }

                }

                else
                {
                        ranAct.writeChars(text.substring(0, fixedSize));

                }

        }

        public static void showRecords() throws IOException
        {

                long no_ofrecords=   ranAct.length()/REC_SIZE;

                ranAct.seek(0);

                for(int i=0; i<no_ofrecords; i++)
                {

                        long actno= ranAct.readLong();

                        String name= readString(NAME_SIZE);

                        float balance= ranAct.readFloat();

                        System.out.println(" "+actno  + " " +name + " "+balance);
                }
        }

        public static String readString (int fixedSize) throws IOException
        {

                String value=" ";

                for(int i=0; i<fixedSize; i++)
                {
                        value= value+ ranAct.readChar();
                }

                return value;
        }
}
```

Note: Here Record Size= 42 (8+30+4)

Suppose that a text file contains marks for 6 courses for a student in a line. Each course marks is separated by space as delimiter. File contains marks for 'n' number of students in separate lines. Write a program that reads the marks from the file for each student and displays the total and average. Your program should prompt the user to enter a filename.

file2.txt contains

15 12 13 16 12 19
13 14 17 18 21 33
12 11 6 7 13 15 19